IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	: Gilbert Wolrich et al.	Art Unit	: 2183
Serial No.	: 10/069,306	Examiner	: Daniel H. Pan
Filed	: July 3, 2002	Conf. No.	: 1999
Title	: INSTRUCTION FOR MULTITHREADED PARALLEL PROCESSOR		

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

BRIEF ON APPEAL

Please charge the brief fee of $510 and please apply any other charges or credits to Deposit Account No. 06-1050.

CERTIFICATE OF MAILING BY EFS-WEB FILING

I hereby certify that this paper was filed with the Patent and Trademark Office using the EFS-WEB system on this date: November 26, 2007

(1) Real Party in Interest

The real party in interest in the above application is Intel Corporation.

(2) Related Appeals and Interferences

The appellant is not aware of any appeals or interferences related to the above-identified patent application.

(3) Status of Claims

This is an appeal from the decision of the Primary Examiner in a Final Office Action dated April 26, 2007. Claims 1, 4-8, 10-16, 18-22, and 24 are pending in the application. Of these, claims 1, 8, 11, 15, 16, 20, 22, 24, and 26 were rejected. Claims 4-7, 10, 12-14, 18, 19, and 21 were objected to as being dependent upon a rejected base claim, but would be allowable if rewritten in independent form including all of the limitations of the base claim and any intervening claims. Claims 2, 3, 9, 17, 23, and 25 have been cancelled.

Claims 1, 8, 11, 15, 16, 20, 22, 24, and 26 are for the subject of this appeal.

(4) Status of Amendments

Appellant filed a Reply to the Non-Final Office Action of October 13, 2006, amending claims 1, 4-8, 10-13, 15, 18-20, 22, and 24, and adding new claim 26. Appellant did not file a Reply to the Final Action. Therefore, all amendments have been entered. Appellant filed a Notice of Appeal on July 26, 2007.

(5) Summary of Claimed Subject Matter

Claim 1

One aspect of Appellant's invention is set out in claim 1 as a method of operating a multithreaded parallel processor. *"...a communication system 10 includes a parallel, hardware-based multithreaded processor 12."*[1]

[1] Specification., page 2, lines 6-7.

Inventive features of claim 1 include directing the processor to have a plurality of microengines to swap, based on a user-specified parameter specified in a context-swap instruction, a currently running context, corresponding to a first thread, in a specified microengine to let another context, corresponding to a different thread that is ready to execute, execute in that microengine and cause a different context and associated program counter to be selected, with the swapped first thread automatically re-enabled to run at some subsequent context arbitration point. *"By employing hardware context swapping within each of the microengines 22a-22f, the hardware context swapping enables other contexts with unique program counters to execute in that same microengine. Thus, another thread e.g., Thread_1 can function while the first thread, e.g., Thread_0, is awaiting the read data to return."*[2] *"The context swap instruction CTX_ARB swaps a currently running context in a specified microengine out to memory to let another context execute in that microengine...The format for the context swap instruction is: ctx_arb[parameter], optional_token"*[3] *"If the thread is swapped, it is automatically re-enabled to run at some subsequent context arbitration point."*[4]

Inventive features of claim 1 also include waking up the swapped out context when the user-specified parameter specified in the context-swap instruction is activated, with the user-specified parameter specifying an occurrence of an event. *"The context swap instruction CTX_ARB also wakes up the swapped out context when a specified signal is activated."*[5] *" The 'parameter' field can have one of several values. If the parameter is specified as 'sram Swap', the context swap instruction will swap out the current context and wake it up when the thread's SRAM signal is received."*[6]

Claim 15

Another aspect of Appellant's invention is set out in claim 15 as a method of operating a multithreaded parallel processor. This feature finds support as the analogous feature of claim 1.

---

[2] Id., page 3, lines 24-28.
[3] Id., page 18, line 31 – page 19, line 3.
[4] Id., page 19, lines 17-18.
[5] Id., page 18, line 33 – page 19, line 1.
[6] Id., page 19, lines 4-6.

Inventive features of claim 15 include receiving a user-specified parameter specified in a context-swap instruction. *"The format for the context swap instruction is: ctx_arb[parameter], optional_token"*[7]

Inventive features of claim 15 also include performing a swapping operation to cause an executing context process corresponding to a first thread to be swapped with a different context and associated program counter, corresponding to a different thread that is ready to execute, the swapped first thread being automatically re-enabled to run at some subsequent context arbitration point. This feature finds support as the analogous feature of claim 1.

Inventive features of claim 15 also include waking up the swapped out context when the user-specified parameter specified in the context-swap instruction is activated, with the user-specified parameter specifying an occurrence of an event. This feature finds support as the analogous feature of claim 1.

Claim 22

Another aspect of Appellant's invention is set out in claim 22 as a parallel processor that can execute multiple contexts. *"...a communication system 10 includes a parallel, hardware-based multithreaded processor 12."*[8] *"The hardware-based multithreaded processor 12...includes a plurality of function microengines 22a-22f...a corresponding plurality of sets of threads can be simultaneously active on each of the microengines 22a-22f while only one is actually operating at a single time."*[9] *"By employing hardware context swapping within each of the microengines 22a-22f, the hardware context swapping enables other contexts with unique program counters to execute in that same microengine."*[10]

Inventive features of claim 22 include a register stack, a program counter for each executing context, and an arithmetic logic unit coupled to the register stack and a program control store that stores a context swap instruction. *"The microengine 22f also includes controller logic 72. The controller logic includes an instruction decoder 73 and program*

---

[7] Id., page 19, lines 1-3.
[8] Id., page 2, lines 6-7.
[9] Id., page 2, lines 27-32.
[10] Id., page 3, lines 24-26

counter (PC) units 72a-72d."[11] *"The microengine 22f also includes an execution box (EBOX) data path 76 that includes an arithmetic logic unit 76a and general purpose register set 76b. The arithmetic logic unit 76a performs arithmetic and logical functions as well as shift functions."*[12] *"The microengine 22f also includes a write transfer register stack 78 and a read transfer stack 80."*[13] *"Both transfer register banks 78 and 80 are connected to the execution box (EBOX) 76 through a data path."*[14] *" a context swap instruction is a special form of a branch that causes a different context (and associated PC) to be selected."*[15]

Inventive features of claim 22 also include the context swap instruction causing the processor to receive a user-specified parameter specified in the context swap instruction. This feature finds support as the analogous feature of claim 1.

Inventive features of claim 22 also include the context swap instruction causing the processor to perform a swap operation to cause an executing context process corresponding to a first thread to be swapped with a different context and associated program counter, corresponding to a different thread that is ready to execute, the swapped first thread is automatically re-enabled to run at some subsequent context arbitration point. This feature finds support as the analogous feature of claim 1.

Inventive features of claim 22 also include the context swap instruction causing the processor to wake up the swapped out context when the user-specified parameter specified in the context-swap instruction is activated, the user-specified parameter specifying an occurrence of an event. This feature finds support as the analogous feature of claim 1.

Claim 24

Another aspect of Appellant's invention is set out in claim 24 as a computer program product residing on a computer readable storage device for causing a multithreaded parallel processor to perform a function. *"The microengine includes a control store 70 which, in one*

---

[11] Id., page 8, lines 31-33.
[12] Id., page 9, lines 20-23.
[13] Id., page 9, lines 28-29.
[14] Id., page 10, lines 2-3.
[15] Id., page 17, lines 21-22.

implementation, includes a RAM of here 1,024 words of 32 bit. The RAM stores a microprogram. The microprogram is loadable by the core processor 20."[16]

Inventive features of claim 24 include the function as receiving a user-specified parameter specified in a context-swap instruction. This feature finds support as the analogous feature of claim 1.

Inventive features of claim 24 also include the function as performing a swapping operation to cause an executing context process corresponding to a first thread to be swapped with a different context and associated program counter, corresponding to a different thread that is ready to execute, the swapped first thread is automatically re-enabled to run at some subsequent context arbitration point. This feature finds support as the analogous feature of claim 1.

Inventive features of claim 24 also include the function as waking up the swapped out context when the user-specified parameter specified in the context-swap instruction is activated, with the user-specified parameter specifying an occurrence of an event. This feature finds support as the analogous feature of claim 1.

(6) Grounds of Rejection to be Reviewed on Appeal

(1) Claim 24 stands rejected under 35 U.S.C. 101 as directed to non-statutory subject matter.

(2) Claims 1, 8, 15, 16, 20, 22, 24, and 26 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Moller (US 4,868,735) in view of Bitar (US 5,872,963) in view of Adkins (US 5,247,671), and Colleran et al. (US 6,005,575).

(3) Claim 11 stands rejected under 35 U.S.C. 103(a) as being unpatentable over Moller in view of Bitar in view of Adkins, and Colleran, and further in view of Turner (US 6,505,229).

(7) Argument

[16] Id., page 8, lines 29-31.

"It is well established that the burden is on the PTO to establish a prima facie showing of obviousness, *In re Fritsch*, 972 F.2d. 1260, 23 U.S.P.Q.2d 1780 (C.C.P.A., 1972)."

In *KSR International Co. v. Teleflex Inc.*, 550 U.S. __, (2007), the Supreme Court reversed a decision by the Court of Appeal's for the Federal Circuit decision that reversed a summary judgment of obviousness on the ground that the district court had not adequately identified a motivation to combine two prior art references. The invention was a combination of a prior art repositionable gas pedal, with prior art electronic (rather than mechanical cable) gas pedal position sensing. The Court first rejected the "rigid" teaching suggestion motivation (TSM) requirement applied by the Federal Circuit, since the Court's obviousness decisions had all advocated a "flexible" and "functional" approach that cautioned against "granting a patent based on the combination of elements found in the prior art."

With respect to the genesis of the TSM requirement, the Court noted that although "As is clear from cases such as Adams[17], a patent composed of several elements is not proved obvious merely by demonstrating that each of its elements was, independently, known in the prior art. Although common sense directs one to look with care at a patent application that claims as innovation the combination of two known devices according to their established functions, it can be important to identify a reason that would have prompted a person of ordinary skill in the relevant field to combine the elements in the way the claimed new invention does. This is so because inventions in most, if not all, instances rely upon building blocks long since uncovered, and claimed discoveries almost of necessity will be combinations of what, in some sense, is already known."

In application of the TSM requirement, the Court cautioned that: "Helpful insights, however, need not become rigid and mandatory formulas; and when it is so applied, the TSM test is incompatible with our precedents."

"The mere fact that the prior art could be so modified would not have made the modification obvious unless the prior art suggested the desirability of the modification." *In re Gordon*, 221 U.S.P.Q. 1125, 1127 (Fed. Cir. 1984).

---

[17] United States v. Adams, 383 U. S. 39, 40 (1966)

> **Although the Commissioner suggests that [the structure in the primary prior art reference] could readily be modified to form the [claimed] structure, "[t]he mere fact that the prior art could be so modified would not have made the modification obvious unless the prior art suggested the desirability of the modification." *In re Laskowski*, 10 U.S.P.Q. 2d 1397, 1398 (Fed. Cir. 1989).**

"The claimed invention must be considered as a whole, and the question is whether there is something in the prior art as a whole to suggest the desirability, and thus the obviousness, of making the combination." *Lindemann Maschinenfabrik GMBH v. American Hoist & Derrick*, 221 U.S.P.Q. 481, 488 (Fed. Cir. 1984).

> **Obviousness cannot be established by combining the teachings of the prior art to produce the claimed invention, absent some teaching or suggestion supporting the combination. Under Section 103, teachings of references can be combined only if there is some suggestion or incentive to do so. *ACS Hospital Systems, Inc. v. Montefiore Hospital*, 221 U.S.P.Q. 929, 933 (Fed. Cir. 1984) (emphasis in original, footnotes omitted).**

"The critical inquiry is whether 'there is something in the prior art as a whole to suggest the desirability, and thus the obviousness, of making the combination.'" *Fromson v. Advance Offset Plate, Inc.*, 225 U.S.P.Q. 26, 31 (Fed. Cir. 1985).

(1) Claim 24 is directed to statutory subject matter within the meaning of 35 U.S.C. § 101.

Claim 24 is directed to a computer program product residing on a computer readable storage device for causing a multithreaded parallel processor to perform a function.

The examiner contends that

> **[a]s...claim 24 as recited is directed to a computer program product residing in a computer readable storage device for causing a multithreaded parallel processor to perform a function (see cliam [sic] 24, preamble). No specific program product components of the program product, nor the specific storage device elements are being reflected into the claim to impart the functionalities of the computer. The details of parallel processor and the computer are not being reflected into the claim. Therefore, the program product, the parallel processor, and the readable storage device are read as general arrangement of the functions al parts, it is read as a general functionality of an arrangement of the generic parts. Furthermore, the**

> **focus is not on the step or feature taken to achieve a final result which is useful, concrete, and tangible, but rather than a final result achieved which is useful, concrete, and tangible. The final result of the method steps, such as the receive, performing, and wakeup, is unclear. Therefore, no substantial practical application can be found.**[18]

Appellant's independent claim 24 recites a "computer readable storage device" which is a tangible element. The examiner argues that: "**No specific program product components of the program product, nor the specific storage device elements are being reflected into the claim to impart the functionalities of the computer.**" However, the examiner overlooks that the computer readable storage device on which the computer program resides includes instructions to impart functionality to a multithreaded parallel processor. The instructions stored on the medium cause a multi-threaded processor, which is also a tangible device, to perform certain functions as recited in the claim. Particularly, the instructions cause the processor to, among other things perform a swap operation that results in the context of a first thread being swapped out, and the context of a second thread swapped in. The second thread then begins execution on the processor. The recited operations are concrete, useful and tangible actions This is sufficient to make the claim statutory.

(2) Claims 1, 8, 11, 15, 16, 20, 22, 24, and 26 are not unpatentable over Moller and Bitar, Adkins, Colleran, and/or Turner.

Claim 1

For the purposes of this appeal only, claims 1, 8, 11, 15, 16, 20, 22, 24, and 26 stand or fall together. Claim 1 is representative of this group of claims.

Claim 1 is directed to a method of operating a multithreaded parallel processor. Moller, Bitar, Adkins, Colleran, and/or Turner, whether taken individually or in any combination, neither describe nor suggest at least the features of: "directing the processor having a plurality of microengines to swap, based on a user-specified parameter specified in a context-swap instruction ... wherein directing the processor comprises waking up the swapped out context

[18] Final Office Action dated April 27, 2007, Paragraph 16, Pages 4-5.

when the user-specified parameter specified in the context-swap instruction is activated, with the user-specified parameter specifying an occurrence of an event."

The examiner contends that

> **Moller ...taught at least...evaluating the state of a parameter of context process corresponding to a first loop (see the generation of CMUXCTL and STKMUXCTL in col.30, lines 39-54, see the invoking loop)[19]**
>
> **[a]s to the newly amended feature of "user-specified", no level of user has been reflected into the claim. Therefore, "user" is read as any user. Moiler [sic] taught his microinstructions were designed to support structures microprogramming and provided the micro program controller with such structured constructs, such as CASE, REPEAT,...UNTIL. DO...WHILE, FOR...DOWN TO, IF...THEN statements (see col.3, lines 44-54). Therefore, the conditions or the arguments were the user specified parameters.[20]**

Applicant disagrees. Applicant contends that Moller discloses a SWAP microinstruction.[21] However, Moller explains the SWAP microinstruction as:

> **FIG. 1 illustrates the typical microprocessor system architecture used in a programmable digital device and can be divided into two distinct sections: a control and instruction acquisition and processing section A on the left and a data acquisition and manipulation section B on the right. Section A has, as its heart, the microprogram sequence controller of the present invention, generally designated with the reference numeral 10. *** Each "machine" instruction is implemented on the microprocessor by a sequence of microinstructions selected by the microprogram sequence controller 10.**
>
> ********
>
> **Microprogram sequence controller 10 generates addresses that determine the sequence of microinstructions that ultimately issue from the microprogram memory 30. The addresses generated by the microprogram sequence controller 10 are conducted from DATA.sub.-- OUT terminals of controller 10 on a "Y" address bus 32 to the address circuits (not shown) of the microprogram memory 30. With the application of such microinstruction addresses, the microprogram memory 30 will issue, on the**

[19] Non-Final Office Action dated October 13, 2006, Paragraph 5, Page 3.
[20] Id., Paragraph 4, page 2.
[21] Moller, Column 30, Line 40.

> **microinstruction bus 34, microinstructions, usually of a word length on the order of 32 or more bits. [22]**

Thus, a microinstruction is but a single step in the execution of one (1) machine instruction. The microinstruction configures the microprocessor in a particular way (e.g., activating a bus or some control signal). Moller's microinstructions, including the SWAP microinstruction, are neither user instructions and nor do the instructions include user-specified parameters. Indeed, microinstructions, such as those described in Moller, do not include any parameters since those microinstructions are hard-wired control sequences designed to electrically connect the processor in some specific pre-determined manner. Moller neither describes nor would it be suggested to include any parameters with the SWAP microinstruction. Moller, therefore, fails to disclose or suggest at least the feature of "directing the processor ... to swap, based on a user-specified parameter specified in a context-swap instruction, ... wherein directing the processor comprises waking up the swapped out context when the user-specified parameter specified in the context-swap instruction is activated, with the user-specified parameter specifying an occurrence of an event," as required by Appellant's independent claim 1.

The examiner relies on Bitar to teach threads contending that

> **Moller did not specifically teach the use of the threads as claimed. Instead, Moller taught the use of loops. However, Bitar disclosed that a suitable number of threads were used to execute a loop in parallel (see col.10, lines 51-59). Therefore, it would have been obvious to one of ordinary skill in the art to use Bitar in Moller for including the threads as claimed because the use of Bitar could provide Moller the ability of the system to adapt to specific processing structure at a predefined set of software constructs (e.g. threads), and thereby, minimizing the overall latency of the loop processing in Moller, and because Moller also taught his system was used as building blocks in an architecture divided in control subsections (see col.1 , lines 24-35), which was a suggestion of the need for providing subdivided control programs for specific subsections of the system, and therefore, one of ordinary skill in the art should be able to recognize the use of threads as taught by Bitar into Moller in order to**

[22] Id., Column 4, Line 33, - Column 5, Line 3. Emphasis added.

achieve the enhanced processing cycle, and in doing so, provided a motivation.[23]

Specifically, the examiner argues: "**However, Bitar disclosed that a suitable number of threads were used to execute a loop in parallel (see col. 10, lines 51-59).**" Appellant contends that Bitar describes a system and method for context switching between a first and a second execution entity (such as a thread) without having to enter into protected kernel mode[24]. Bitar also describes an exemplary procedure for effecting a context switch between two threads,[25] part of which may be implemented using a sequence of assembly instructions.

Appellant disagrees. While Bitar does discuss threads, and context switching, Bitar does not describe a context-swap instruction that causes a context switch nor does Bitar describe a context swap instruction with a user-specified parameter, indicative of an occurrence of an event used to wake up a swapped-out context. Therefore, any combination of Moller with Bitar still neither describes nor suggests at least the features of "directing the processor having a plurality of microengines to swap, based on a user-specified parameter specified in a context-swap instruction, ... wherein directing the processor comprises waking up the swapped out context when the user-specified parameter specified in the context-swap instruction is activated, with the user-specified parameter specifying an occurrence of an event," as required by Appellant's independent claim 1.

The examiner relies on Adkins to teach "**a wakeup of a context (see col.8, lines 45-64).**" Appellant disagrees. Appellant contends that Adkins does not teach the feature as claimed, but instead describes a scheduler executing on a serial communications adapter that schedules tasks at different priority levels.[26] Adkins describes that a sleeping task can be awakened and have its context restored when a new event associated with that task arrives at the port response queue[27].

Specifically, the examiner contends that

---

[23] Non-Final Office Action dated October 13, 2006, Paragraph 8, Pages 4-5.
[24] Bitar Abstract
[25] Id. FIG. 9, and Column 13, Line 35, to Column 14, Line 64.
[26] Adkins, Abstract.
[27] Id., Column 8, Lines 36-57.

> **[n]either Moller nor Bitar specifically showed the wakeup of the swap out context as claimed. However, Adkins disclosed a wakeup of a context (see col.8, lines 45-64). It would have been obvious to one of ordinary skill in the ad to use Adkins in Moller for including the wakeup of the swapped context as claimed because the use of Adkins could provide Moller the control ability to restore the context of the associated task at a predetermined cycle, thereby reducing the latency caused during the resume of the task process, and it could be readily achieved by predefining the wakeup signals of Adkins into the swap instructions of Moller with modified read and write operands so that the specific wakeup command of the context of a given task of Adkins could be recognized by Moller, and because Moller did teach the activation of the context on the top of a stack by a selection signal caused by the swap instruction (see col.30, lines 39-44), which was a suggestion of the need of a function specific signal (e.g. wakeup, sleep, etc) into the swap instruction in order to achieve the enhanced context recovery or activation, and for the above reasons , provided a motivation. Moller is used as a primary reference because it clearly showed a swap instruction (see swap instruction) in col.30, lines 39-40). Adkins is used as secondary because it supplemented the teaching of the wakeup of the context.**[28]

Adkins does not describe use of program instructions to control swapping, nor does Adkins describe user-specified parameters indicative of an occurrence of an event used to wake-up a swapped out thread. Specifically Adkins does not disclose the feature of "waking up the swapped out context when the user-specified parameter specified in the context-swap instruction is activated." Therefore, Adkins does not disclose or suggest at least the features of "directing the processor having a plurality of microengines to swap, based on a user-specified parameter specified in a context-swap instruction, ... wherein directing the processor comprises waking up the swapped out context when the user-specified parameter specified in the context-swap instruction is activated, with the user-specified parameter specifying an occurrence of an event," as required by Appellant's independent claim 1.

The examiner also acknowledges that **"Moller did not specifically teach the automatically enabled first thread as claimed. However, Colleran taught automatically re-enable of a thread (see the automatic enabling of the thread by the process in col.5, lines 62-67, col.6, lines 1-24)."** Appellant disagrees. Colleran describes a method and system for foreground window determination of windows displaying applications in a displayed desktop.[29] The examiner contends that

[28] Non-Final Office Action dated October 13, 2006, Paragraph 9, Pages 5-6.
[29] Colleran, Abstract.

> **Moller did not specifically teach the automatically enabled first thread as claimed. However, Colleran taught automatically re-enable of a thread (see the automatic enabling of the thread by the process in col.5, lines 62-67, col.6, lines 1-24). It would have been obvious to one of ordinary skill in the art to use Colleran in Moller for including the automatically re-enabled first thread as claimed because the use of Colleran could provide Moller the ability to restart the task at desirable point of processing, and it could be readily achieved by predefining the automatic enabled thread of Colleran into the configuration file of Moller with modified system parameters (e.g. the enable flag and the R/W flags) such that the automatic enablement of Colleran's thread could be recognized by Moller, and because Moller also taught an activation of the context on the top of a stack by a selection signal caused by the swap instruction (see col.30, lines 39-44), which was a suggestion of the need for automatically activating or enabling the specific task or thread at system command, therefore increasing the system level flexibility, and in doing so, provided a motivation.**[30]

While Colleran describes threads and automatically initiating Appellant is unable to find where Colleran teaches "the swapped first thread automatically re-enabled to run at some subsequent context arbitration point," the feature that is actually claimed in claim 1. Appellant contends that Colleran neither describes nor suggests a context-swap instruction, to effect context switching. Therefore, Colleran does not address the claimed feature that the examiner relies on Colleran to teach and moreover does not cure any of the deficiencies in any combination of Moller, Bitar, Adkins. Colleran also does not describe use of a user-specified parameter, indicative of an occurrence of an event to wake-up a swapped-out thread, that is used in conjunction with a context swap instruction. Accordingly, Colleran does not disclose or suggest at least the features of "directing the processor having a plurality of microengines to swap, based on a user-specified parameter specified in a context-swap instruction, ... wherein directing the processor comprises waking up the swapped out context when the user-specified parameter specified in the context-swap instruction is activated, with the user-specified parameter specifying an occurrence of an event," as required by Appellant's independent claim 1.Because none of the references cited by the examiner discloses or suggests, alone or in combination, at least the features of "directing the processor having a plurality of microengines to swap, based on a user-specified parameter specified in a context-swap instruction, ... wherein

[30] Non-Final Office Action dated October 13, 2006, Paragraph 10, Page 6.

directing the processor comprises waking up the swapped out context when the user-specified parameter specified in the context-swap instruction is activated, with the user-specified parameter specifying an occurrence of an event," Appellant's independent claim 1 is patentable over the cited art.

Claims 8, 26 depend from independent claim 1 and are therefore patentable for at least the same reasons as independent claim 1.

Independent claims 15, 22 and 24 recite "receiving a user-specified parameter specified in a context-swap instruction; performing a swapping operation to cause an executing context process corresponding to a first thread to be swapped with a different context and associated program counter, corresponding to a different thread that is ready to execute, the swapped first thread being automatically re-enabled to run at some subsequent context arbitration point; and waking up the swapped out context when the user-specified parameter specified in the context-swap instruction is activated, with the user-specified parameter specifying an occurrence of an event," or similar language. For reasons similar to those provided with respect to independent claim 1, at least these features are not disclosed by the cited art. Appellant's independent claims 15, 22 and 24 are therefore patentable over the cited art.

Claims 16 and 20 depend from independent claim 15 and are therefore patentable for at least the same reasons as independent claim 15.

(3) Claim 11 is patentable over Moller, Bitar, Adkins, Colleran, and Turner.

Claim 11 further limits claim 1, and requires that "the user-specified parameter specifies "start_receive" which swaps out the current context and wakes up the swapped, current context when new data in a receive FIFO is available for this thread to process."

The examiner acknowledges that the base combination of references as applied to claim 1 do not suggest this feature. Accordingly the examiner relies on Turner. Specifically the examiner states that, in

> **...claim 11 , neither Moller nor Bitar, nor Adkins, nor Colleran specifically showed the available data in the receive FIFO as claimed. However, Turner disclosed a system including a ready FIFO queue for allowing a swapping for a thread (see col.7, lines 38-37 [sic]). It would have been obvious to one of ordinary skill in the ad [sic] to use Turner in Moller for including the available data in FIFO as claimed because the use of Turner could provide additional capability of the given swap command to initiate the swapping at the available cycle of the FIFO , therefore, minimizing the wait time in the access cycle, and it could be readily done by configuring the FIFO of Turner with modified interface ports into Moller so that the available data in FIFO of Turner could be recognized by Moller, and for the above reasons , provided a motivation.**[31]

Claim 11 is further distinct over Turner. While Turner teaches a "**...particular method of handling threads purely on a time-slice basis...and a FIFO next-thread Ready queue [that] allows for very efficient task swapping...**"[32], that is, swapping out a context and waking up the swapped, current context when new data in a receive FIFO is available for this thread to process, nowhere does Turner disclose or suggest that such a context-swapping be initiated via a user-specified parameter as required by Appellant's independent claim 1. Instead, context-swapping in Turner is initiated through a time-slicing mechanism, in which "**[t]he variable time-slice classification is determined upon thread creation...and is fixed for the life of the thread...variable time-slicing is constrained by heredity and privilege for threads created by other threads.**"[33] Turner's time-slicing mechanism that controls context-swapping is not defined by a user-defined parameter, but is automatically determined through pre-determined conditions. Therefore, Turner does not address the claimed feature that the examiner relies on Turner to teach and moreover does not cure any of the deficiencies in any combination of Moller, Bitar, Adkins, or Colleran. Appellant's claim 11 is patentable over the cited art.

---

[31] Non-Final Office Action dated October 13, 2006, Paragraph 13, Page 7.
[32] Turner, Column 7, Lines 52-57.
[33] Id., Column 7, Lines 4-8.

Conclusion

Appellant submits, therefore, that claims 1, 8, 11, 15, 16, 20, 22, 24, and 26 are neither described by nor obvious over any purported combination of Moller, Bitar, Adkins, Colleran, and/or Turner and are otherwise allowable over the cited art. Therefore, the Examiner erred in rejecting Appellant's claims and should be reversed.

Respectfully submitted,

Date: 11/26/07

Denis G. Maloney
Reg. No. 29,670

Fish & Richardson P.C.
225 Franklin Street
Boston, MA 02110
Telephone: (617) 542-5070
Facsimile: (617) 542-8906

21778166.doc

Appendix of Claims

1. A method of operating a multithreaded parallel processor comprising:

directing the processor having a plurality of microengines to swap, based on a user-specified parameter specified in a context-swap instruction, a currently running context, corresponding to a first thread, in a specified microengine to let another context, corresponding to a different thread that is ready to execute, execute in that microengine and cause a different context and associated program counter to be selected, with the swapped first thread automatically re-enabled to run at some subsequent context arbitration point,

wherein directing the processor comprises waking up the swapped out context when the user-specified parameter specified in the context-swap instruction is activated, with the user-specified parameter specifying an occurrence of an event.

2-3. (Cancelled)

4. The method of claim 1 wherein the user-specified parameter specifies "sram Swap", which swaps out the current context and wakes up the swapped, current context when the thread's SRAM signal is received.

5. The method of claim 1 wherein the user-specified parameter specifies "sdram Swap," which swaps out the current context and wakes up the swapped, current context when the thread's SDRAM signal is received.

6. The method of claim 1 wherein the user-specified parameter specifies "FBI" which swaps out the current context and wakes up the swapped, current context when the thread's FBI signal is received indicating that an FBI CSR, Scratchpad, TFIFO, or RFIFO operation has completed.

7. The method of claim 1 wherein the user-specified parameter specifies "seq_num1_change/seq_num2_change", which swaps out the current context and wakes up the swapped, current context when a value of the sequence number changes.

8. The method of claim 1 wherein the user-specified parameter specifies "inter_thread" which swaps out the current context and wakes up the swapped, current context when the thread's interthread signal is received.

9. (Cancelled)

10. The method of claim 1 wherein the user-specified parameter specifies "auto_push" which swaps out the current context and wakes up the swapped, current context when SRAM transfer read register data has been automatically pushed by a FBus interface.

11. The method of claim 1 wherein the user-specified parameter specifies "start_receive" which swaps out the current context and wakes up the swapped, current context when new data in a receive FIFO is available for this thread to process.

12. The method of claim 1 wherein the user-specified parameter specifies "kill" which prevents the current context or thread from executing again until an appropriate enable bit for the thread is set in a CTX_ENABLES register.

13. The method of claim 1 wherein the user-specified parameter specifies "pci" which swaps out the current context and wakes up the swapped, current context when a PCI unit signals that a DMA transfer has been completed.

14. The method of claim 1 wherein directing further comprises:

in response to an optional_token "defer one" specified in the context-swap instruction, executing an additional instruction in an instruction stream of the currently running context before the context is swapped.

15. A method of operating a multithreaded parallel processor, the method comprising:

receiving a user-specified parameter specified in a context-swap instruction;

performing a swapping operation to cause an executing context process corresponding to a first thread to be swapped with a different context and associated program counter, corresponding to a different thread that is ready to execute, the swapped first thread being automatically re-enabled to run at some subsequent context arbitration point; and

waking up the swapped out context when the user-specified parameter specified in the context-swap instruction is activated, with the user-specified parameter specifying an occurrence of an event.

16. The method of claim 15 wherein performing comprises swapping a currently running context in a specified microengine to let another context execute in that microengine.

17. (Cancelled)

18. The method of claim 15 wherein the user-specified parameter specifies "sram Swap", and performing a swapping comprises swapping out the current context and waking up the swapped, current context when the thread's SRAM signal is received.

19. The method of claim 15 wherein the user-specified parameter specifies "sram Swap", and performing a swapping comprises swapping the current context and waking up the swapped, current context when the thread's SDRAM signal is received.

20. The method of claim 15 wherein the user-specified parameter specifies "inter_thread" which swaps out the current context and wakes up the swapped, current context when the thread's interthread signal is received.

21. The method of claim 15 further comprising:

in response to an optional_token "defer one" specified in the context-swap instruction, executing an additional instruction in an instruction stream of the currently running context before the context is swapped.

22. A parallel processor that can execute multiple contexts and that comprises:

a register stack;

a program counter for each executing context;

an arithmetic logic unit coupled to the register stack and a program control store that stores a context swap instruction that causes the processor to:

receive a user-specified parameter specified in the context swap instruction;

perform a swap operation to cause an executing context process corresponding to a first thread to be swapped with a different context and associated program counter, corresponding to a different thread that is ready to execute, the swapped first thread is automatically re-enabled to run at some subsequent context arbitration point; and

wake up the swapped out context when the user-specified parameter specified in the context-swap instruction is activated, the user-specified parameter specifying an occurrence of an event.

23. (Cancelled)

24. A computer program product residing on a computer readable storage device for causing a multithreaded parallel processor to perform a function, the computer program product comprising instructions causing the processor to:

receive a user-specified parameter specified in a context-swap instruction;

perform a swapping operation to cause an executing context process corresponding to a first thread to be swapped with a different context and associated program counter, corresponding to a different thread that is ready to execute, the swapped first thread is automatically re-enabled to run at some subsequent context arbitration point; and

wake up the swapped out context when the user-specified parameter specified in the context-swap instruction is activated, with the user-specified parameter specifying an occurrence of an event.

25. (Cancelled)

26. The method of claim 1 wherein the user-specified parameter specifies "voluntary".

Evidence Appendix

None

Related Proceedings Appendix

None